

# Secure hierarchy-based access control in distributed environments\*

Jean-Camille Birget<sup>△</sup>, Xukai Zou<sup>†</sup>, Guevara Noubir<sup>⊕</sup> and Byrav Ramamurthy<sup>†</sup>

<sup>△</sup>Dept. of Computer Science, Rutgers University, Camden, USA

<sup>†</sup>Dept. of Computer Science and Engineering, University of Nebraska-Lincoln, USA

<sup>⊕</sup>College of Computer Science, Northeastern University, USA

E-mail: birget@camden.rutgers.edu, {xkzou, byrav}@cse.unl.edu, noubir@ccs.neu.edu

February 28, 2002

## Abstract

Access control is a fundamental concern in any system that manages resources, e.g., operating systems, file systems, databases and communications systems. The problem we address is how to specify, enforce, and implement access control in distributed environments.

Starting from an access relation between users and resources, we derive a user hierarchy, a resource hierarchy, and a unified hierarchy. The unified hierarchy is then used to specify the access relation in a way that is compact and that allows efficient queries. It is also used in cryptographic schemes that enforce access rights. We introduce three specific cryptography-based hierarchical access schemes, which can effectively enforce and implement access control and are designed for distributed environments.

**Keywords:** Distributed access control, access hierarchies, information and communication security.

---

\*Note: This work was conducted when G. Noubir and J.-C. Birget were visiting the University of Nebraska-Lincoln. J.-C. Birget was supported in part by NSF grant DMS-9970471 and B. Ramamurthy was supported in part by NSF grant EPS-0091900 and by UNL MoCoRePro Project. An earlier version of this paper appeared in the Proceedings of the IEEE International Conference on Communications (ICC), 2001 [4].

# 1 Introduction

The subject domain of this paper is distributed applications in environments such as distributed operating systems, distributed database systems, and communication networks, where different users access various resources with different access rights. This subject is called distributed access control. Other examples include management of distributed project resources, web subscription services, pay TV, etc.

To elaborate further, in the context of management of project resources, users include directors, group leaders, project managers, technical managers, engineers, consultants, administrative staff, customers and accounting staff. Resources include financial data, internal technical documents, public project documents, laboratories, etc. Different users have different access rights to different resources, which need to be concisely specified and correctly enforced.

Access control deals with the specification and enforcement of users' access permissions relative to the resources of a system [2, 7, 13]. This is a fundamental concern in any system that manages resources. Traditionally, access control is specified by an *access relation* (or "access matrix") that lists explicitly which users can access which resources.

In this paper we uncover a user hierarchy and a resource hierarchy, that are implicit in any access relation. Intuitively the hierarchies arise from the fact that some users have more access rights than others, and some resources carry more access restrictions than others (a formal definition will be given later). We show that these hierarchies can give useful information.

Another contribution of this paper is an algorithm that merges these user and resource hierarchies into a single hierarchy. This unified hierarchy contains the user and resource hierarchy as sub-hierarchies; moreover, a user is above a resource in the unified hierarchy if and only if this user has access to this resource. Thus the unified hierarchy contains all the information of the access relation, while also displaying the useful hierarchy information. In addition, the unified hierarchy merges 'equivalent' users, and merges 'equivalent' resources (rigorous definitions will be given); thus the unified hierarchy will usually be a compact description of the access rights.

Having a unified hierarchy can simplify access control. The literature contains a number of secure access control protocols [1, 5, 9, 11, 14] that assume (without justification) that users and resources are organized in pre-existing hierarchies (see Subsection 3.4). We show that these secure access control schemes can use our unified hierarchy in order to enforce access permissions and restrictions.

For a centralized system, access control is usually implemented by a centrally stored access table [3, 8, 10, 15, 16].

However, applications in distributed environments call for distributed access control. In this paper, we provide access control schemes that are specifically designed for distributed applications.

In this paper we do not consider the dynamics of access control (when users and resources are added and removed and when access rights change). Our results are applicable when systems change only slowly. Dynamical distributed access control is a very difficult problem, which is studied in a more restricted domain in [17] (e.g., for tree-hierarchies and secure group communications).

In the next section we define the user and resource hierarchies, as well as the unified hierarchy, and prove the existence and uniqueness of the unified hierarchy. In Section 3, we discuss the specification of an access relation (in particular, using the unified hierarchy) and introduce three cryptography-based schemes which enforce the access relation; these schemes use the unified hierarchy and are specifically designed for distributed applications.

## 2 Hierarchies that are implicit in an access relation

A *hierarchy* is formalized by a directed acyclic graph (a “DAG”), which defines a partial order (“hierarchical order”) among vertices. A vertex  $v_i$  is below a vertex  $v_j$  in the hierarchical order ( $v_i \leq v_j$ ) if and only if there exists a directed path in the graph from  $v_j$  to  $v_i$ .

Let  $U = \{u_1, u_2, \dots\}$  be the set of **users** in the system, and let  $R = \{r_1, r_2, \dots\}$  be the set of **resources** in the system. The **access relation**  $\mathbf{A}$  of the system determines which resources each user can legally access and use:

$$\mathbf{A} = \{(u, r) \in U \times R : \text{the user } u \text{ can access the resource } r\}.$$

For a user  $u \in U$ , let  $R(u) \subseteq R$  denote the set of resources that  $u$  can access; for a resource  $r \in R$ , let  $U(r) \subseteq U$  denote the set of users that can access  $r$ . So,  $(u, r) \in \mathbf{A}$  is equivalent to  $r \in R(u)$ , and also equivalent to  $u \in U(r)$ . In the following, we assume that the access relation is completely known, and hence all the sets  $R(u)$  and  $U(r)$  are known.

The user and resource hierarchies are defined as follows:

**Definition 2.1.** *Let  $u_i, u_j \in U$ ,  $r_i, r_j \in R$ .*

- $u_i \leq_U u_j$  if and only if  $R(u_i) \subseteq R(u_j)$ . In words,  $u_j$  is above  $u_i$  in the user hierarchy if and only if every resource that  $u_i$  can access can also be accessed by  $u_j$ .

- $r_i \leq_R r_j$  if and only if  $U(r_j) \subseteq U(r_i)$ . In words,  $r_i$  is below  $r_j$  in the resource hierarchy if and only if every user that can access  $r_j$  can also access  $r_i$ .
- $u_i \equiv_U u_j$  if and only if  $R(u_i) = R(u_j)$ . In words, two users are equivalent (regarding access control) if and only if they can access exactly the same resources (i.e., they have the same access rights).
- $r_i \equiv_R r_j$  if and only if  $U(r_i) = U(r_j)$ . In words, two resources are equivalent (regarding access control) if and only if they are accessible by exactly the same users.

Notice that the subset order is reversed for resources, compared to users: Users with more access rights are higher in the user hierarchy, whereas resources with more access rights are lower in the resource hierarchy. This is natural, once one thinks of the meaning of the access hierarchy.

The ‘order’ relations defined so far are in general not anti-symmetric (i.e.,  $x \leq y$  and  $x \geq y$  does not always imply  $x = y$ ; see e.g., [6] for the mathematical definition of ‘partial order’). To obtain partial orders we **merge equivalent** ( $\equiv_U$ ) **users** into single groups, and we **merge equivalent** ( $\equiv_R$ ) **resources** into single groups. Note that the result is the same, whether we first merge equivalent users, and then equivalent resources, or vice-versa. From now on, when we say “user” (or “resource”), we will mean a group of equivalent users (respectively, resources).

Now (after merging equivalent users, and merging equivalent resources), the users form a partial order (“p.o.”), called the **user hierarchy**, and denoted by  $(U, \leq_U)$ ; similarly, the resources form a p.o., called the **resource hierarchy**, and denoted by  $(R, \leq_R)$ .

**Example:** We illustrate the hierarchies by the following simple example, inspired from a college environment (Figure 1). The the set of users is  $\{\text{prof1, prof2, grStu1, grStu2, ugrStu1, ugrStu2, \dots, ugrStu100, secr, sysMgr, sysHelp}\}$  (i.e., two professors, two graduate students, a hundred undergraduates, a secretary and two systems people); the set of resources is  $\{\text{c1, c1A, c2, c3, lab1, lab2, pr1, pr2}\}$  (i.e., four class rooms, two labs, and two printers). The access relation is given in Figure 1, which describes the users’ adjacency lists (i.e., for each user there is one row which lists the resources that are accessible by this user).

Figures 2 and 3 represent the user and the resource hierarchies, determined by the access relation according to Definition 2.1. Notice that the users grStu1 and grStu2 have access to exactly the same resources, so we have merged grStu1 and grStu2; from now on the group  $\{\text{grStu1, grStu2}\}$  is treated as a single user. In a similar way,  $\{\text{ugrSt1, \dots}$

```

prof1 → c1, c1A, c3, lab1, lab2, pr1, pr2
prof2 → c2, c3, lab1, lab2, pr1, pr2
grStu1 → c1A, c3, lab1, lab2, pr2
grStu2 → c1A, c3, lab1, lab2, pr2
ugrStu1 → c3, lab1, lab2, pr2
ugrStu2 → c3, lab1, lab2, pr2
...
ugrStu100 → c3, lab1, lab2, pr2
secr → c3, pr1, pr2
sysMgr → all resources
sysHelp → all resources except c1 and c2

```

Figure 1: Example of an access relation

, ugrSt100} forms a group of users with the same access rights, and will be treated as a single user. Also, the resources lab1 and lab2 are accessible by the same users, so they are merged into the group {lab1, lab2}; that group will now be viewed as a single resource. Similarly, the resources c3 and pr2 are merged.

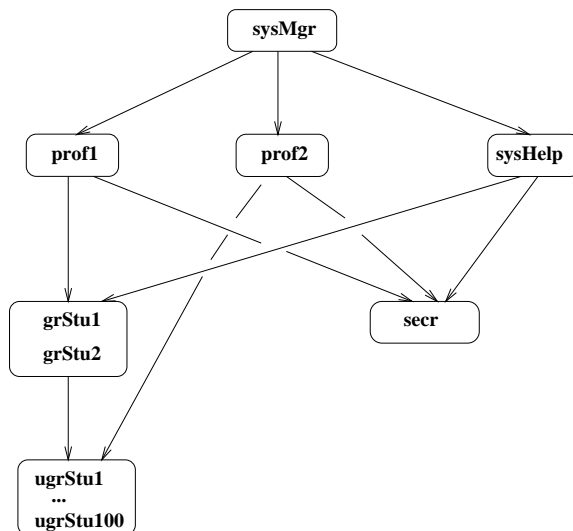


Figure 2: User hierarchy

It may look strange, at first, that we merge different users into groups, and similarly for different resources. Merged

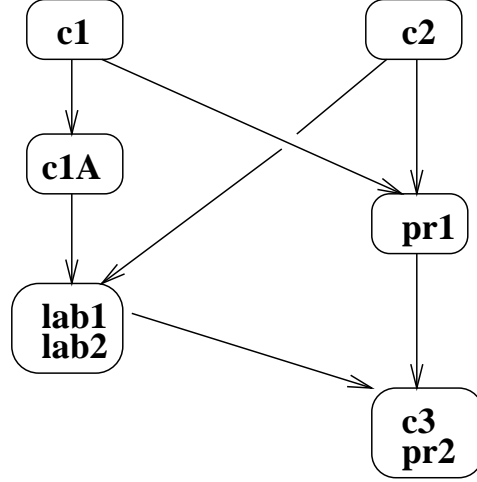


Figure 3: Resource hierarchy

entities could be quite different in nature, but equivalence means that *with respect to access control* the merged entities need not be distinguished. It may also seem strange that we order entities according to their access rights. An individual with a high rank (e.g., in an organization’s pecking order) might be fairly low in the user hierarchy of Definition 2.1. However, for access control, the hierarchies introduced here are the relevant ones.

We will now combine the user hierarchy and the resource hierarchy into a unified hierarchy. Before constructing a uniform hierarchy, let us define what we mean by that.

**Definition 2.2.** Let  $(U, \leq_U)$  and  $(R, \leq_R)$  be p.o.’s (user hierarchy and resource hierarchy, respectively), obtained from an access relation **A**. The **unified hierarchy** is a partial order (“p.o.”)  $(V, \preceq)$  satisfying the following conditions:

- (1) The user hierarchy is a sub-p.o. of the unified hierarchy. This means:  $(U, \leq_U)$  is embedded into the p.o.  $(V, \preceq)$  by a one-to-one map  $f_U : U \rightarrow V$ , such that for all  $u_i, u_j \in U$ :  $u_i \leq_U u_j$  if and only if  $f_U(u_i) \preceq f_U(u_j)$ .
- (2) Similarly, the resource hierarchy is a sub-p.o. of the unified hierarchy. This means:  $(R, \leq_R)$  is embedded into the p.o.  $(V, \preceq)$  by a one-to-one map  $f_R : R \rightarrow V$ , such that for all  $r_i, r_j \in R$ :  $r_i \leq_R r_j$  if and only if  $f_R(r_i) \preceq f_R(r_j)$ .
- (3) A user  $u \in U$  has access to a resource  $r \in R$  if and only if  $u$  is above  $r$  in the unified hierarchy (i.e., if and only if  $f_R(r) \preceq f_U(u)$ ).
- (4) In the p.o.  $(V, \preceq)$  no mergers of elements are possible, unless some of the conditions (1), (2), or (3) are violated.

The following theorem shows that a unified hierarchy, as just defined, exists and is not larger than the combined

size of the two original user and resource hierarchies. We will use the notation  $x \succeq y$  to mean  $y \preceq x$ .

**Theorem 2.3.** *Let  $\mathbf{A} \subseteq U \times R$  be any access relation, and let  $(U, \leq_U)$  and  $(R, \leq_R)$  be the user, respectively the resource hierarchy determined by  $\mathbf{A}$ . Then there **exists** a unified hierarchy  $(V, \preceq)$  (satisfying the conditions of Definition 2.2), and this unified hierarchy is **unique** up to ‘isomorphism’ (i.e., up to renaming the elements of  $V$ ).*

*$(V, \preceq)$  is the smallest p.o. (regarding the size of  $V$ ), satisfying conditions (1), (2), and (3) of Definition 2.2. For the size  $|V|$  of  $V$  we have:  $|V| \leq |U| + |R|$ .*

*Moreover,  $(V, \preceq)$  can be constructed from  $\mathbf{A}$  in polynomial time.*

**Proof.** First, in Lemma 2.4 we construct a hierarchy  $(V, \preceq)$  which satisfies conditions (1), (2), and (3), and  $|V| \leq |U| + |R|$ . The construction is actually an algorithm, which runs in polynomial time.

Note that in a unified hierarchy, no two users can be merged (by condition (1) of Definition 2.2), and no two resources can be merged (by condition (2) of Definition 2.2). However, a user could sometimes be merged with a resource in the unified hierarchy; Lemma 2.5 describes when that happens. Minimality and uniqueness of  $(V, \preceq)$  follow immediately from Lemma 2.5.  $\square$

**Lemma 2.4.** *Let  $\mathbf{A}$ ,  $(U, \leq_U)$  and  $(R, \leq_R)$  be as in Theorem 2.3. Then there exists a hierarchy  $(V, \preceq)$  which satisfies conditions (1), (2), and (3) of Definition 2.2. For the size  $|V|$  of  $V$  we have:  $|V| \leq |U| + |R|$ .*

*Moreover,  $(V, \preceq)$  can be constructed from  $\mathbf{A}$  in polynomial time.*

**Proof.** We use the classical notation  $2^R$  for the set of all subsets of  $R$ . We will construct the unified hierarchy  $(V, \preceq)$  as a sub-p.o. of the p.o.  $(2^R, \subseteq)$ . (We could have based the construction on  $2^U$ , which would have been quite similar.)

(1) We embed the user hierarchy  $(U, \leq_U)$  into  $(2^R, \subseteq)$  by the map

$$f_U : u_i \in U \mapsto f_U(u_i) = R(u_i) \in 2^R.$$

In words,  $f_U(u_i)$  is the set of resources that  $u_i$  can access. Then  $f_U$  is one-to-one (because we already merged equivalent users). Moreover,  $u_i \leq_U u_j$  if and only if  $f_U(u_i) \subseteq f_U(u_j)$ , by the very definition of  $\leq_U$ .

(2) We embed the resource hierarchy  $(R, \leq_R)$  into the p.o.  $(2^R, \subseteq)$  by the map

$$f_R : r_i \in R \mapsto f_R(r_i) = \{r_j \in R : r_j \leq_R r_i\}.$$

In words,  $f_R(r_i)$  is the set of resources that are below  $r_i$  in the resource hierarchy. Then  $f_R$  is one-to-one by anti-symmetry of the p.o.  $\leq_R$  (after merger of equivalent resources). Moreover,  $r_i \leq_R r_j$  if and only if  $f_R(r_i) \subseteq f_R(r_j)$ , by transitivity of  $\leq_R$ .

(3) The third condition of the definition holds:

$u_i$  can access  $r_j$  if and only if  $r_j \in R(u_i)$ ;

this is true if and only if for all  $r_k \leq_R r_j : r_k \in R(u_i)$ ;

and this is equivalent to saying  $f_R(r_j) \subseteq f_U(u_i)$ .

Let  $V = \{f_U(u) : u \in U\} \cup \{f_R(r) : r \in R\} \subseteq 2^R$ . Then the p.o.  $(V, \subseteq)$  satisfies conditions (1), (2), (3) of Definition 2.2 (with  $\subseteq$  playing the role of  $\preceq$ ). Also, clearly  $|V| \leq |U| + |R|$ .

It is easy to implement the construction of  $(V, \preceq)$  in polynomial time; note that we need not consider all of  $2^R$  in the construction but only the  $|V| (\leq |U| + |R|)$  sets mentioned above). This proves Lemma 2.4.  $\square$

**Lemma 2.5.** *Let  $(W, \preceq)$  be any unified hierarchy obtained from  $\mathbf{A}$ ,  $(U, \leq_U)$  and  $(R, \leq_R)$ , and satisfying conditions (1), (2), (3) of Definition 2.2, with embedding maps  $F_U$  and  $F_R$ . Then*

(a)  $F_U(u_i) = F_R(r_j)$  implies  $R(u_i) = \{r_k : r_k \leq_R r_j\}$ .

If, in addition,  $(W, \preceq)$  satisfies condition (4) of Definition 2.2 then we have

(b)  $F_U(u_i) = F_R(r_j)$  if and only if  $R(u_i) = \{r_k : r_k \leq_R r_j\}$ .

Moreover, in that case  $(W, \preceq)$  is isomorphic to the p.o.  $(V, \preceq)$  constructed in Lemma 2.4 (i.e., the two p.o.'s differ only in the names of their elements).

Note that the last line in the Lemma implies that  $(V, \preceq)$  satisfies condition (4) of Definition 2.2. It also follows that  $V$  has minimum size (indeed, a minimum-size unified hierarchy satisfies (4), hence, by the Lemma, is isomorphic to  $(V, \preceq)$ ). And it follows that the unified hierarchy (satisfying (1) - (4) is unique, since all hierarchies satisfying (1) - (4) are isomorphic to  $(V, \preceq)$ .

**Proof.** (a) Assume  $F_U(u_i) = F_R(r_j)$ . By condition (3) of Definition 2.2, this implies that  $u_i$  can access  $r_j$  (and hence  $u_i$  can also access the descendants of  $r_j$ ); thus we have  $\{r_k : r_k \leq_R r_j\} \subseteq R(u_i)$ .

Conversely let  $r_h$  be any resource in  $R(u_i)$  (i.e.,  $u_i$  can access  $r_h$ ). Then (by condition (3) of Definition 2.2), we have  $F_U(u_i) \succeq F_R(r_h)$ . Hence (since we assume  $F_U(u_i) = F_R(r_j)$ ),  $F_R(r_j) \succeq F_R(r_h)$ . Condition (2) of Definition

2.2 then implies,  $r_j \geq_R r_h$ , i.e.,  $r_h \in \{r_k : r_k \leq_R r_j\}$ . So, we have  $\{r_k : r_k \leq_R r_j\} = R(u_i)$ .

(b) As a consequence, we have a well-defined map  $W \rightarrow V$  (where  $V$  was constructed in Lemma 2.4), defined by  $F_U(u_i) \mapsto f_U(u_i) = R(u_i)$  and  $F_R(r_j) \mapsto f_R(r_j) = \{r_k : r_k \leq_R r_j\}$ . The map is well defined since  $F_U(u_i) = F_R(r_j)$  implies  $\{r_k : r_k \leq_R r_j\} = R(u_i)$  (as we just proved).

Since the p.o.  $(W, \preceq)$  maps onto the p.o.  $(V, \preceq)$ , we have the following, if  $(W, \preceq)$  satisfies condition (4) of Definition 2.2:  $F_U(u_i) = f_U(u_i) = f_R(r_j) = F_R(r_j)$ . This proves the Lemma.  $\square$

Henceforth we will simply write  $u$  and  $r$  for elements in the unified hierarchy (instead of  $f_U(u)$  or  $f_R(r)$ ).

The definition of the unified hierarchy does not tell us explicitly what it means for a resource to be above a user ( $u \preceq r$ ). From the construction one can derive the following.

**Proposition 2.6.** *In the unified hierarchy the following are equivalent (where  $u$  is a user and  $r$  is a resource):*

- (1)  $u \preceq r$ ;
- (2) every resource accessible by  $u$  is  $\leq_R r$ ;
- (3) every user who can access  $r$  is  $\geq_U u$ ;
- (4) for every user  $u_i$  who can access  $r$ , and every resource  $r_j$  which is accessible by  $u$  we have:

$u_i$  can access  $r_j$ ;

- (5) the Cartesian product  $U(r) \times R(u)$  is a subset of  $\mathbf{A}$ .

**Proof.** [(1)  $\Leftrightarrow$  (2)] By definition,  $f_U(u) \preceq f_R(r)$  if and only if  $R(u) \subseteq \{r_j : r_j \leq_R r\}$ . Putting this into words amounts to saying that every resource accessible by  $u$  is  $\leq_R r$ .

[(2)  $\Leftrightarrow$  (5)] Property (2) can be written as  $(\forall r_j \in R(u)) : r_j \leq_R r$ .

This is equivalent to  $(\forall r_j \in R(u)) : U(r) \subseteq U(r_j)$

(since “ $r_j \leq_R r$ ” was defined to mean “ $U(r) \subseteq U(r_j)$ ”).

This is equivalent to  $(\forall r_j \in R(u))(\forall u_i \in U(r)) : u_i \in U(r_j)$ .

This is equivalent to  $(\forall r_j \in R(u))(\forall u_i \in U(r)) : (u_i, r_j) \in \mathbf{A}$ .

This is equivalent to  $U(r) \times R(u) \subseteq \mathbf{A}$ .

[(3)  $\Leftrightarrow$  (5)] Property (3) can be written as  $(\forall u_i \in U(r)) : u_i \geq_U u$ .

This is equivalent to  $(\forall u_i \in U(r)) : R(u) \subseteq R(u_i)$

(since “ $u_i \geq_U u$ ” was defined to mean “ $R(u) \subseteq R(u_i)$ ”).

This is equivalent to  $(\forall u_i \in U(r))(\forall r_j \in R(u)) : r_j \in R(u_i)$ .

This is equivalent to  $(\forall u_i \in U(r))(\forall r_j \in R(u)) : (u_i, r_j) \in \mathbf{A}$ .

This is equivalent to  $U(r) \times R(u) \subseteq \mathbf{A}$ .

[(5)  $\Leftrightarrow$  (4)] Putting property (5) in words amounts to property (4).  $\square$

**Example:** For the example of Figure 1, Figure 4 represents the unified hierarchy. Note that some users have been merged with resources. One can check that the resources accessible by prof1 are exactly the resources below  $(\leq_R)$  c1 in the resource hierarchy; according to Lemma 2.5 this is the criterion for merging a user and a resource. Similarly, the user group {grStu1, grStu2} has been merged with the resource c1A, because the resources accessible to {grStu1, grStu2} are exactly the resources  $\leq_R$  c1A.

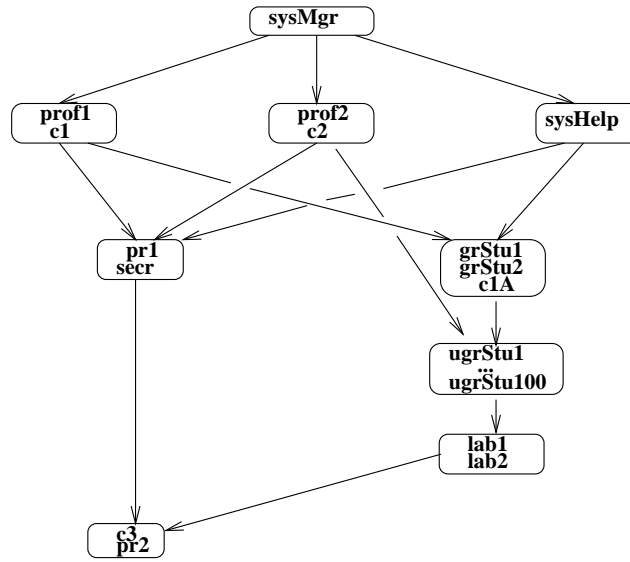


Figure 4: Unified hierarchy

### 3 Implementation of access control

In this section we use the hierarchies that we introduced in order to develop secure access control schemes. Hierarchies can be used for both the specification and the enforcement of access control. In our enforcement schemes, a user  $u_i$  has to prove to a resource  $r_j$  that  $u_i$  has the right to access  $r_j$ , and this should be possible if and only if  $r_j \preceq u_i$  with respect

to the unified hierarchy. We will present three basic means to enforce access control: certificates, unconditionally secure keying schemes, and computationally secure keying schemes (based on one-way functions).

### 3.1 Specification of access control

An access control relation can be given explicitly, by an *access matrix*, which can be useful for theoretical reasonings but is wasteful for space. A more compact description of the access control relation can be given by *adjacency lists*: user-dominant adjacency lists or resource-dominant adjacency lists. A user-dominant adjacency list consists of one row per user; the row of that user mentions the user and all the resources that this user can access. Similarly, a resource-dominant adjacency list consists of one row per resource, and the row of that resource mentions the resource and all the users that have access to this resource.

The *unified hierarchy* can also be used to describe the access relation. In this case, the hierarchy is given as a graph in which every vertex is labeled by the set of equivalent users and/or resources that are represented by this vertex. Because of the merger of equivalent users or resources, and the merger of some users with some resources, the unified hierarchy is a representation which is as compact as (and usually more compact than) the adjacency list representation. Moreover, the unified hierarchy has the advantage that certain queries are more efficient: Given a user  $u_i$  or a resource  $r_j$  it is easy to find the adjacency list of  $u_i$  or  $r_j$  (namely, pick all the resources that are  $\preceq u_i$ , respectively, all the users that are  $\succeq r_j$ ). In the user-dominant adjacency list representation, it is tedious to find a resource's adjacency list; on the other hand, if both user- and resource-dominant adjacency lists are explicitly given, storage is wasted. Thus, the unified hierarchy could serve as a representation of the access relation, which is both compact and efficient for queries.

Various mixed representations of the access relation are also possible: We might be given partial information about adjacency lists, about the user and resource hierarchies, or information about equivalence of some users or some resources. This may arise in specifications, and one could be asked to reconstruct the entire unified hierarchy from these data.

In a distributed environment, partial information about the access relation or the unified hierarchy will be distributed among the users and the resources; no central authority is needed (except may be at the set-up of the system or for occasional maintenance and updates).

In the next three subsections we give schemes for enforcing an access relation.

### 3.2 Certificate-based schemes

In these schemes a trusted certificate authority (CA) distributes certificates to users. When a user accesses a resource the protocol is as follows: The user provides an access request along with a certificate. The resource then verifies the user's access right based on this certificate (without consulting the CA).

It is natural to assume that users know which resources they can access. A user  $u_i$  may have a certificate of the following form for each resource  $r_j$  that  $u_i$  can access:

[  $u_i$ 's ID,  $(u_i, r_j)$ , cert.-valid-time, CA-sig. ].

Here,  $u_i$ 's ID identifies the user,  $(u_i, r_j)$  indicates the access right, and the CA's digital signature certifies to the resource that the information in the certificate is correct. We refer to books on cryptography for more information on certificates and digital signatures (e.g., [12]).

Alternatively, instead of having a different certificate for each resource that  $u_i$  can access,  $u_i$  might have just one certificate that lists all of  $R(u_i)$  (i.e., all the resources accessible to  $u_i$ ). This approach may be simpler when the number of resources is small; but it gives more information to a resource than this resource needs to know.

In any case, no information about the access relation or the hierarchies needs to be stored in the resources.

A disadvantage of this scheme comes from a general problem with certificates: it is hard to keep certificates up to date when the system changes (the certificate revocation problem – see [12]).

### 3.3 Unconditionally secure keying schemes

Let  $(V, \preceq)$  be a unified hierarchy. In these keying schemes, every vertex  $v$  in the unified hierarchy has a key  $k_v$  (picked from a large key space). Each user at  $v$  knows a set of keys  $U_v \subseteq \{k_w : w \preceq v\}$  (i.e., the user knows some keys of lower-ranking resources), and each resource at vertex  $v$  knows a set of keys  $R_v \subseteq \{k_w : v \preceq w\}$  (i.e., the resource knows some keys of higher-ranking users). The sets  $U_v$  and  $R_v$  must be chosen in such a way that

$$(*) \quad v_i \preceq v_j \text{ if and only if } U_{v_j} \cap R_{v_i} \neq \emptyset.$$

A user  $u_j$  (at vertex  $v_j$ ) requesting a resource  $r_i$  (at vertex  $v_i$ ) will present the set  $U_{v_j}$  to the resource. The resource then checks whether  $U_{v_j} \cap R_{v_i} \neq \emptyset$ , which holds if and only if  $v_i \preceq v_j$ , i.e., if and only if  $u_j$  has the right to access  $r_i$ .

Note that equivalent users receive the same keying material, which is fine since they have the same access rights.

In this protocol, a resource can get information about the keying material held by users. A user  $u_j$ , however, only knows his set of keys  $U_{v_j}$  and no other keys. An alternative would be to use (local) trusted third parties, to whom keying material is presented by users and resources; the trusted third parties do not need to remember the sets  $U_v$  and  $R_v$ ; they just have to check condition (\*) when a user and a resource present their sets.

There are many ways of choosing the sets  $U_v$  and  $R_v$  so that condition (\*) holds. We will illustrate this approach by simple special cases, namely ‘user multiple keying’, ‘resource multiple keying’, and ‘mixed keying’.

**(a) User multiple keying:**

In this scheme the set of keys are as follows. For every vertex  $v$ :

$$R_v = \{k_v\}, \text{ and}$$

$$U_v = \{k_{v_j} : v_j \preceq v\}.$$

A user  $u$  requesting access to a resource  $r$  presents  $U_u$  to  $r$ . The resource  $r$  verifies  $u$ 's access right by checking whether  $k_r \in U_u$  (which is equivalent to condition (\*) in this case).

Example: See Figure 5, which represents two copies of a unified hierarchy; the first copy gives the sets  $R_v$ , the second copy gives the sets  $U_v$ . Here,  $U_{v_2} = \{k_2, k_4, k_5\}$ ,  $R_{v_2} = \{k_2\}$ ,  $R_{v_4} = \{k_4\}$ ,  $R_{v_5} = \{k_5\}$ , therefore user  $u_2$  can access resources  $r_2, r_4$ , and  $r_5$ .

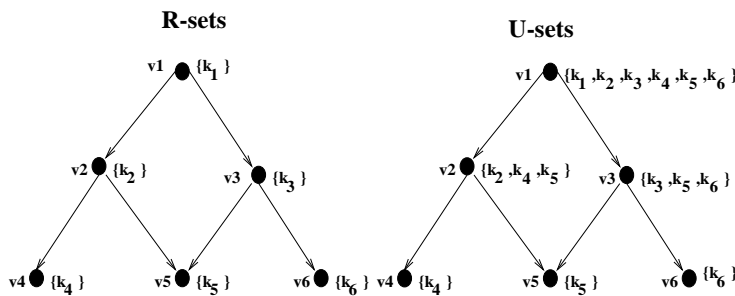


Figure 5: User multiple keying

**(b) Resource multiple keying:**

This scheme is similar to User multiple keying, with the roles of user and resources switched.

**(c) Mixed keying:**

This is the general case. There are many possibilities, and we give just one example (see Figure 6): Here,  $U_{v_1} = \{k_1, k_5\}$ ; moreover,  $R_{v_2}, R_{v_3}, R_{v_4}, R_{v_6}$  contain the key  $k_1$ , so any user  $u_1$  at vertex  $v_1$  can access the resources at vertices  $v_2, v_3, v_4, v_6$ . User  $u_1$  can also access the resource at  $v_5$  because  $U_{v_1}$  and  $R_{v_5}$  share the key  $k_5$ . One can check from the two graphs in Figure 5 that condition (\*) holds at all the vertices.

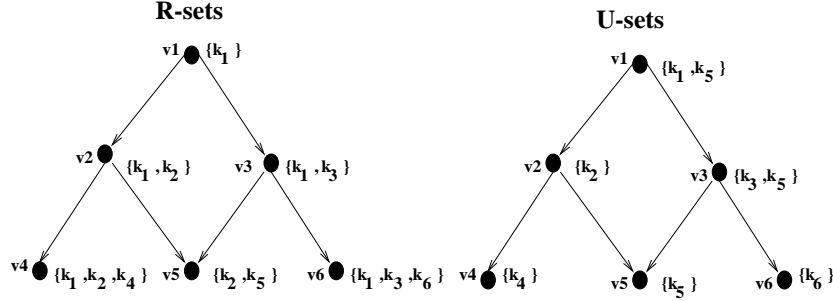


Figure 6: Mixed-Keying

### 3.4 One-way function based keying schemes

A drawback of the unconditionally secure schemes presented above is that the total amount of secret information that the users or the resources have to store can be quite large (proportional to the number of vertices in the unified hierarchy). One-way functions can drastically improve this. We present a way of using one-way functions, namely a generalization of Lin’s scheme [11], based on the unified hierarchy. Other methods (e.g., the scheme of Akl and Taylor [1]) could be generalized as well by using the unified hierarchy.

#### A generalization of Lin’s scheme

Lin’s original scheme [11] assumed that the user hierarchy and the resource hierarchy were the same. However, we do not need this assumption; we will simply apply Lin’s method to our unified hierarchy  $(V, \preceq)$ . One of the advantages of the unified hierarchy is that methods like Lin’s scheme now work without special assumptions.

Moreover, we will use any one-way function  $F : \mathbf{K} \times \mathbf{I} \rightarrow \mathbf{K}$ , where  $\mathbf{K}$  is a large key space, and  $\mathbf{I}$  is the space of vertex identifiers. (We let  $\mathbf{I}$  be a large set of possible names for vertices, such that  $V \subset \mathbf{I}$  for any particular vertex set  $V$  that we will use.) Both  $\mathbf{K}$  and  $\mathbf{I}$  are sets of bit strings.

Every vertex  $v \in V$  is assigned its own independent key  $k_v \in \mathbf{K}$ . Only  $v$  (and no other vertex) knows  $k_v$ . The access relation is represented by the unified hierarchy, which itself will be represented by an array of numbers  $r_{vw}$

( $v, w \in V$ ), defined as follows (where  $\oplus$  denotes bitwise exclusive OR):

$$r_{vw} = F(k_w, v) \oplus k_v, \quad \text{if } v \preceq w;$$

$r_{vw}$  is a random element of  $\mathbf{K}$  if  $v \not\preceq w$ .

In any case, the elements  $(r_{vw}, v, w)$  (as  $v$  and  $w$  range over  $V$ ) are made public. The one-way function  $F$  is also assumed to be publicly known.

Now, if  $v \preceq w$  then  $w$  can compute  $k_v (= F(k_w, v) \oplus r_{vw})$ , using  $k_w$  (which  $w$  knows) and  $r_{vw}$  (which is public). On the other hand, if  $v \not\preceq w$ , the element  $r_{vw}$  is random and carries no information (and therefore,  $k_v$  cannot be derived from  $k_w$  and  $r_{vw}$ ). A user associated with vertex  $w$  accesses a resource associated with vertex  $v$  by presenting  $k_v$  (computed from  $k_w$  and  $r_{vw}$ , as we saw) to  $v$ . (Lin used a special one-way function, which required some set-up work. However, we do not need the special properties of Lin's one-way function; any good one-way function will do.)

The advantages of this scheme are that every vertex  $v$  can select its own key  $k_v$  in a decentralized manner; moreover, a vertex does not need to remember the entire hierarchy. The hierarchy is represented by the numbers  $r_{vw}$  which are public, and could be stored in a publicly accessible (not necessarily secure) data base.

## 4 Conclusion

We showed that three hierarchies can be extracted from an access relation: a user hierarchy, a resource hierarchy, and a unified hierarchy. Moreover, we can define an equivalence relation among users (and among resources) that merges users (respectively resources) that are equivalent for access control. These hierarchies allow compact specifications of access control, and are useful for schemes that enforce an access relation. Cryptographic key-based hierarchical schemes can be designed to effectively enforce and implement access control in distributed environments.

The issue of dynamic access control remains a challenging problem for future research.

## References

- [1] S.G. Akl, P.D. Taylor, "Cryptographic solution to a problem of access control in a hierarchy", *ACM Transactions on Computer Systems* 1(3) (1983) 239-247.

- [2] E. Amoroso, *Fundamentals of Computer Security Technology*, Prentice-Hall International, Englewood Cliffs, NJ, 1994.
- [3] E. Bertino, S. Jajodia, P. Samarati, "A flexible authorization mechanism for relational data management systems", *ACM Transactions on Information Systems*, 17(2) (1999) 101-140.
- [4] J.C. Birget, X. Zou, G. Noubir, B. Ramamurthy, "Hierarchy-based access control in distributed environments", *International Conference on Communications ICC 2001*, Helsinki, Finland (June 2001); paper NGI6.8
- [5] G.C. Chick, S.E. Tavares, "Flexible access control with master keys", *Advances in Cryptology: CRYPTO '89*, LNCS Vol. 435, Springer-Verlag, (1990) 316-322.
- [6] B.A. Davey, H.A. Priestley, *Introduction to Lattices and Order*. Cambridge University Press (1990).
- [7] D.E. Denning. *Cryptography and Security*. Addison-Wesley, Reading, MA, 1982.
- [8] H.M. Gladney, "Access control for large collections", *ACM Transactions on Information Systems* 15(2) (1997) 154-194.
- [9] S.J. Greenwald, "A new policy for distributed resource management and access control", *Proceedings of the UCLA Conference on New Security Paradigms Workshops* (1996) 74-86.
- [10] T. Jaeger, A. Prakash, J. Liedtke, N. Islam, "Flexible control of downloaded executable content", *ACM Transactions on Information and Systems Security*, 2(2) (1999) 177-228.
- [11] C.H. Lin, "Dynamic key management schemes for access control in a hierarchy", *Computer Communications* 20 (1997) 1381-1385.
- [12] A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, Boca Raton, Florida (1996).
- [13] C.P. Pfleger. *Security in Computing*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 1997.
- [14] R.S. Sandhu, "Cryptographic implementation of a tree hierarchy for access control", *Information Processing Letters* 27 (1988) 95-98.

- [15] G. Thomas, G.R. Thompson, C.W. Chung, E. Barkmeyer, F. Carter, M. Templeton, S. Fox, B. Hartman, "Heterogeneous distributed database systems for production use", *ACM Computing Surveys* 22(3) (1990) 237-266.
- [16] E. Wobber, M. Abadi, M. Burrows, B. Lampson, "Authentication in the TAOS operating system", *ACM SIGOPS* (December 1993) 256-269.
- [17] X. Zou, Secure group communications and hierarchical access control, PhD. Thesis, University of Nebraska – Lincoln, USA (December 2000).